COLLECTIONS

1. PROBLEM STATEMENT

XYZ Online Bookstore is revamping its user authentication and account management system to improve security and user experience. As a software developer assigned to this project, you are tasked with implementing a User class and an AccountManager class based on the following requirements:

User Class

- Attributes:
  - `username`: A string representing the username of the user.
  - `password`: A string representing the password of the user.
  - `email`: A string representing the email address of the user.
  - Create getter and setter methods for all attributes.
  - Implement a default constructor that initializes the `username`, `password`, and `email` attributes to empty strings.

AccountManager Class

- Attributes:
  - `userList`: A list to store instances of the User class.
  - Create getter and setter methods for `userList`.
  - Implement a default constructor that initializes `userList` as an empty list.

Requirements:

1. **registerUser(String username, String password, String email):**
  - Method to register a new user.
  - Check if the `username` and `email` are unique. If either is already taken, return `false` and do not register the user.
  - If both `username` and `email` are unique, create a new User instance with the provided details, add it to `userList`, and return `true`.

2. **login(String username, String password):**
  - Method to authenticate a user's login attempt.
  - Check if the `username` exists in `userList` and if the corresponding `password` matches the provided password.
  - If valid, return the User object associated with the username. If not valid, return `null`.

3. **resetPassword(String username, String newPassword):**
  - Method to reset a user's password.
  - Find the user in `userList` by `username` and update the `password` to `newPassword`.
  - Return `true` if the password reset was successful, or `false` if the username was not found in `userList`.

4. **updateEmail(String username, String newEmail)**:
   - Method to update a user's email address.
   - Find the user in `userList` by `username` and update the `email` to `newEmail`.
   - Return `true` if the email update was successful, or `false` if the username was not found in `userList`.

Additional Functionality:
- Implement a method `getUserByUsername(String username)` in AccountManager to retrieve a User object by username from `userList`.
- Implement a method `deleteUser(String username)` in AccountManager to remove a user from `userList` by username.

Main Program Interaction:
- The program interacts with the user as follows:
  - Prompt the user to choose an action (register, login, reset password, update email, delete user).
  - Depending on the action:
    - For registration, prompt for `username`, `password`, and `email`.
    - For login, prompt for `username` and `password` and display whether the login was successful.
    - For password reset, prompt for `username` and `newPassword` and display whether the password was successfully reset.
    - For email update, prompt for `username` and `newEmail` and display whether the email was successfully updated.
    - For user deletion, prompt for `username` and confirm if the user should be deleted from `userList`.

Sample Input/Output:
```
Enter action (register, login, reset password, update email, delete user): register
Enter username: alice
Enter password: pass123
Enter email: alice@example.com
User registered successfully.

Enter action (register, login, reset password, update email, delete user): login
Enter username: alice
Enter password: pass123
Login successful. Welcome, Alice!

Enter action (register, login, reset password, update email, delete user): reset password
Enter username: alice
Enter new password: newpass456
Password reset successful.

Enter action (register, login, reset password, update email, delete user): update email
Enter username: alice
Enter new email: newalice@example.com
```

Email updated successfully.

Enter action (register, login, reset password, update email, delete user): delete user
Enter username: alice
Are you sure you want to delete user 'alice'? (yes/no): yes
User 'alice' deleted successfully.

Enter action (register, login, reset password, update email, delete user): login
Enter username: alice
Enter password: newpass456
Invalid credentials. Please try again.
```

---

==========

## 2. PROBLEM STATEMENT

ABC Electronics is a leading retailer specializing in electronic gadgets. The company wants to enhance its inventory management system to efficiently track sales based on product categories. The IT department has assigned you, a software developer, to implement a program based on the requirements.

**SalesTracker Class**

**salesSet** =>A set to store sales records in the format of `customerName:product` pairs. And create setter and getter and default constructor.

**SalesTracker** => Constructor that initializes an empty `salesSet`.

**Requirement 1: addSalesRecord**(String record) => Method to add a sales record to `salesSet`. Each record is provided in the format `customerName:product`.

**Requirement 2: findNumberOfCustomersByProduct(product)** =>Method to count the number of unique customer names who purchased a specific product. If no customers are found for the given product, return `-1`.

**Requirement 3: getCustomersByProduct(product) =>** Method to retrieve a list of customer names who purchased a specific product. If no customers are found for the given product, return an empty list.

**Requirement 4: getProductByCustomer(customerName) =>** Method to find and return the product purchased by a specific customer. If no product is found for the given customer, return `None`.

In the `**MAIN**` class, the program interacts with the user as follows:

- Prompt the user to enter the number of sales records they want to add.
- Collect the sales records from the user in the format `Customer Name:Product`.
```

- Split the input string by the delimiter and invoke the `addSalesRecord` method to add the sales records. After adding records:
- Prompt the user to enter a product name to find the number of customers who purchased it.
- Display the count of customers or print `"No customers found for <product>"` if no customers are found.
- Prompt the user to enter a product name to retrieve the list of customers who purchased it.
- Display the list of customers or print `"No customers found for <product>"` if no customers are found.
- Prompt the user to enter a customer name to find the product purchased by that customer.
- Display the product purchased by the customer or print `"No product found for <customer name>"` if no product is found.

Sample Input/Output:

Input/Output 1:

Enter number of records to be added

3

Enter the records (Customer Name:Product)

Alice:Laptop

Bob:Monitor

Charlie:Keyboard

Enter the Product to be searched

laptop

The number of customers who purchased laptop is 1

Enter the Product to identify the Customer Names

Monitor

Customer names who purchased Monitor are

Bob

Enter the Customer name to find the Product purchased

Charlie

Product purchased by Charlie is Keyboard

Input/Output 2:

Enter number of records to be added

2

Enter the records (Customer Name:Product)

Eve:Headphones

Frank:Tablet

Enter the Product to be searched

smartphone

No customers found for smartphone

Enter the Product to identify the Customer Names

Headphones

Customer names who purchased Headphones are

Eve

Enter the Customer name to find the Product purchased.

adam

No product found for adam

Input/Output 3:

Enter number of records to be added

4

Enter the records (Customer Name:Product)

Alice:Phone

Bob:TV

Charlie:Speaker

David:Phone

Enter the Product to be searched

TV

The number of customers who purchased TV is 1

Enter the Product to identify the Customer Names

Phone

Customer names who purchased Phone are

Alice, David

Enter the Customer name to find the Product purchased

carl

No product found for carl

==========

3. PROBLEM STATEMENT

Queen Elizabeth II decided to go for the caffeine survey in England, she wants Sarah John to get all the coffee brands present in the UK stores and the average percentage of caffeine present in it.

Sarah being a code enthusiast found it easy to calculate the average value through the codes.

Can you write the code for Sarah?

Your task here is to complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications

Task

Class Caffeine

```
class definitions:
  class Caffeine:
      data member:
       coffeeBrand: String
       caffeinePercentage: float
        visibility: private

   Caffeine(String coffeeBrand, float caffeinePercentage): constructor with pub
lic visibility
    Define getter setters with public visibility

  class Sorting implements Comparator<Caffeine>
       method definitions:
           compare(Caffeine o1, Caffeine o2):
             return type: int
             visibility: public

  class CaffeineSurvey
      data member:
          ArrayList<Caffeine> coffeeList

      method definitions:
         sortByBrandName():
             return type: ArrayList<Caffeine>
             visibility: public

         getAvgPercentage():
             return type: double
             visibility: public
```

- define the **String** variable **coffeeBrand**
- define the **float** variable **caffeinePercentage**
-define a **constructor** and **getter setters** according to the above specifications

Class Sorting

Implement the below methods for this class:

class definitions:
class Caffeine:

data member:
coffeeBrand: String
caffeinePercentage: float
visibility: private

Caffeine(String coffeeBrand, float caffeinePercentage): constructor with public visibility

Define getter setters with public visibility

class Sorting implements Comparator<Caffeine>

method definitions:
compare(Caffeine o1, Caffeine o2):
return type: int
visibility: public

class CaffeineSurvey

data member:
ArrayList<Caffeine> coffeeList
method definitions:
sortByBrandName():
return type: ArrayList<Caffeine>
visibility: public

getAvgPercentage():
return type: double
visibility: public

-int compare(Caffeine o1, Caffeine o2):
• Write a **sorting** logic for sortByBrandName method

Class CaffeineSurvey
- define the **ArrayList<Caffeine>** variable **coffeeList**
Implement the below methods for this class:
-ArrayList<Caffeine> sortByBrandName():
• Write a code to get the **coffeeList sorted** in ascending order by coffeBrand
• Do not change the **coffeeList** to maintain the default order of the list
• Return the new sorted list
-double getAvgPercentage():
• Write a code to to find the average percentage from **coffeeList**.
• Return the average percentage.

Sample Input

```
ArrayList<Caffeine> list = new ArrayList<>();
 list.add(new Caffeine("Nescafe", (float)69.2));
 list.add(new Caffeine("CarteNoir", (float)54.5));
CaffeineSurvey cs = new CaffeineSurvey();
cs.coffeeList = list;
-----------------------------------------------------
cs.sortByBrandName();
cs.getAvgPercentage();
```

Sample Output

```
{"cartenoir", "nescafe"}
61.85
```

==========

4. PROBLEM STATEMENT

Can You Align:

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications:

```
class definitions:
 class Size:
    data members:
        int area
        visibility : private
    Size(int area): constructor with public visibility
    define getter and setter with public visibility

 class Content:
    data members:
        String text
        boolean isImage
         visibility :private
   Content(String text, boolean isImage): Constructor with public visibility
    define getter and setter with public visibility

 class WebPage:
    data members:
      Map<Integer, Map<Size,Content>> page = new HashMap<>();
      Size totalArea
      int index
      visibility : public

     WebPage(Size totalArea): Constructor with public visibility

   method definition:
      addPage(Size size, Content content):
         return : String
         visibility : public

      checkArea(Size size):
         return : boolean
         visibility : public

      canCompress(Content content):
         return : boolean
         visibility : public

      compressSize(Size size):
         return : int
         visibility : public
```

TASK

Class Size

- define all the variables according to the above specifications.

class definitions:

class Size:

data members:

int area
visibility : private
Size(int area): constructor with public visibility
define getter and setter with public visibility

class Content:

data members:

String text
boolean isImage
visibility :private

Content(String text, boolean isImage): Constructor with public visibility

define getter and setter with public visibility

class WebPage:

data members:

Map<Integer, Map<Size,Content>> page = new HashMap<>();
Size totalArea
int index
visibility : public

WebPage(Size totalArea): Constructor with public visibility

method definition:

addPage(Size size, Content content):
return : String
visibility : public

checkArea(Size size):
return : boolean
visibility : public

canCompress(Content content):
return : boolean
visibility : public

compressSize(Size size):
return : int
visibility : public

- define a **constructor with getter and setter** according to the above specifications.

Class Content
- define all the variables according to the above specifications.
- define a **constructor with getter and setter** according to the above specifications.

Class WebPage
- define all the variables according to the above specifications.
- define a **constructor** according to the above specifications.

Implement the below methods for this class:

-String addPage(Size size, Content content):

• Write a code that adds the send parameter to the **HashMap page** on the basis of defined conditions.
• If the given **size** is greater than the totalArea then return **"No space".**
• If the given **size** is equal to or less than the **totalArea** then add it to the HashMap (page) according to the below conditions -

1. If the **canCompress** method returns **true** then add it to the **HashMap(page)** with the size returned by the **compressSize** method and return "**Page added after compress".**
2. If the **canCompress** method returns **false** then add it to the **HashMap(page)** and returns **"Page added".**
3. The initial value of the index is 0 and it got increments after every successful addition to the HashMap(page).

-String checkArea(Size size):

• Write a code that returns true if the totalArea object's area is greater than equal to the size object's area otherwise return false.

-String canCompress(Content content):

• Write a code that returns true if the content's text length is greater than 100 and content's isImage is true else return false.

-String compressSize(Size size):

• Write a code that returns area on the basis of the following condition -
1. If the area is greater than 100 then return 75% of the area.
2. Otherwise returns the are area without modifications.

Sample Input

```
Size totalArea = new Size(150);
Size size = new Size(20);
Content content = new Content("Hi web", false);
WebPage page = new WebPage(totalArea);
page.addPage(size2, content);
Content content1 = new Content("Hi webHi webHi webHi webHi webHi webHi webHi
webHi webHi webHi webHi webHi webHi webHi web", true);
page.addPage(size2, content);
```

Sample Output

```
Page added
Page added after compress
```